AMENDMENTS TO THE SPECIFICATION

Please add the following paragraphs on page 1 of the substitute specification, after the heading Description of Related Technology:

In general, logic optimization is classified into two categories, two-level logic optimization and multi-level logic optimization.

Two-level optimization deals with the optimization of combinational logic circuits, modeled by two-level "sum of products" expression forms, or equivalently by tabular forms such as implicant tables. Two-level logic optimization has a direct impact on programmable logic arrays (PLAs) and macro-cell based programmable logic devices (CPLDs).

Combinational logic circuits are very often implemented as multi-level networks of logic gates. The fine granularity of multi-level networks provides several degrees of freedom in logic design that may be exploited in optimizing area and delay as well as in satisfying specific constraints, such as different timing requirements on different input/output paths. Thus, multi-level networks are very often preferred to two-level logic implementations such as PLAs. The unfortunate drawback of the flexibility in implementing combinational functions as multi-level networks is the difficulty of modeling and optimizing the networks themselves. The need of practical synthesis and optimization algorithms for multi-level circuits has made this topic of high importance in VLSI CAD.

Multi-level logic optimization is frequently partitioned into two steps. In the first step, a logic network is optimized while neglecting the implementation constraints on the logic gates and assuming rough models for their area and performance. This procedure is usually referred to as technology independent logic optimization. In the second step, one takes into consideration the constraints on the available gates (e.g., K-LUTs in FPGAs) as well as the detailed area and delay models of these gates. This step is the so-called technology dependent logic optimization or technology mapping. The discussion hereinbelow addresses the technology independent logic optimization problem, and, in particular, the timing-driven logic resynthesis problem.

Several common operations that are used during the area-oriented multi-level optimization are as follows:

1. Common sub-expression extraction

By extracting common sub-expressions from a number of functions, the circuit area is reduced. However, the better the area saving, the more places the sub-expression fans out to, which could degrade the circuit performance.

2. Resubstitution

Resubstitution is similar to common sub-expression extraction and involves expressing a node in terms of another, if possible.

3. Elimination

Elimination involves removing, from the multi-level network, all occurrences of variables that represent the nodes which are eliminated. When all the internal nodes are eliminated, the operation is called collapsing.

4. Decomposition

The decomposition of an internal node function in a multi-level network replaces the node by two (or more) nodes that form a subnetwork equivalent to the original node. Decomposition is often performed on a node to split a complex function into two (or more) simpler functions. Small-sized expressions are more likely to be divisors of other expressions and may enhance the ability of the resubstitution algorithm to reduce the size of the network.

5. Simplification using don't care conditions

Simplification is used to find a compact representation for the Boolean function at every node. By removing the redundancies from a representation of a function, both the size and the depth can be reduced. In a multi-level network, the simplification at a node needs to consider the structure of the logic around it. This gives rise to don't care conditions that can be exploited during node simplification.

From the description of these operations, one can see the complex interaction between the circuit area and delay. In addition, the delay impact of a particular transformation applied on the same network often depends on the delay data (the arrival and required times). Since the delay data is imprecise at the technology independent stage, it is difficult to adapt the strategies used for area optimization to address the performance optimization issue. Because of this difficulty, many of the techniques developed to reduce the circuit delay use local transformations to make incremental changes to the logic.

Timing optimization will now be discussed. One significant issue in restructuring a circuit is determining circuit regions that should be transformed. The most critical outputs and their transitive fanins are a natural choice. However, one problem with this approach is that after the most critical outputs have been optimized, outputs that were close to being critical before could become critical after optimization of the original critical paths. Moreover, optimizing only the most critical outputs by more than the needed amount can also result in an unnecessary area penalty. Thus, some techniques optimize close-to-critical nodes along with the most critical nodes.

Several conventional algorithms use an iterative refinement-based approach, where, in each iteration, a set of critical paths is identified and then the delays of a set of nodes are reduced so that the overall circuit performance is improved. These algorithms are differentiated in (i) how to determine in each iteration the set of nodes to apply the local transformation for delay reduction and (ii) the local transformation method itself.

Another conventional attempt at timing optimization takes a different approach based on clustering, partial collapsing and subsequent timing optimization. This approach is based on the premise that at a technology-independent level, in the absence of the target technology information and wiring delays, any delay model is inaccurate. Therefore, it assigns a zero delay to all the gates, thus treating all the input-to-output paths uniformly. However, whenever a signal crosses cluster boundaries, a delay of one unit is incurred.

Another existing approach first performs area optimization on a circuit to achieve to reduce the size of the circuit layout, and then incremental changes are made to the circuit to reduce its delay. This approach is particularly useful for layout-driven logic resynthesis, wherein the timing correction step is performed incrementally to ensure the convergence of the iteration between the layout design and the circuit resynthesis.

A significant aim of the restructuring approaches discussed above is to generate a good multi-level structure of the circuit that will subsequently be mapped into a small delay implementation. These conventional approaches generally use simple, weak models to predict the circuit delay. As a result, the savings observed at the technology independent stage may not be evident after technology mapping of the optimized circuit.

To alleviate this problem, researchers have extended the basic ideas of the technology independent optimizations to work on mapped circuits. Heuristics have been used to address the optimization of mapped circuits while taking into account the characteristics of the cell library.

The Timing-Driven Logic Optimization section discussion below describes the performance optimization at the technology independent level and how this optimization impacts the subsequent technology mapping and physical design.

With the rapid scaling of transistor feature sizes, integrated circuit performance is increasingly determined by interconnects instead of devices. Interconnect delays are even more significant in PLD designs due to the extensive use of programmable switches. As a result, the delay between two logic blocks is highly dependent on their placement on the chip and the routing resources used to connect them. PLDs, such as those from Altera, include several types of interconnects, including local, row and column interconnects. Local interconnects refer to the connections between logic elements (LEs) in the same logic array block (LAB). Row interconnects refer to the connections between LEs in the same row, but in different LABs. Column interconnects refer to the connections between LEs in different rows. The delay attributed to interconnects can be many times that of the logic element delay. Given such a high variation of different types of interconnect delays, it would be almost impossible to perform accurate timing optimization during synthesis without proper consideration of the layout result. That is why layout-driven synthesis is considered to be an important problem area in high-performance PLD designs.

The layout-driven synthesis problem has proved to be difficult to solve due to the mutual dependency nature of the logic synthesis and layout design. In general, there are two approaches to integrate logic and layout synthesis. One approach is to employ a highly iterative design flow. It follows the design steps in the traditional design flow, but feeds the layout result in the current iteration back to the logic synthesis tools for improving the synthesis results in the next iteration. To make such a "construct-by-correction" approach effective, the correction step needs to be done incrementally with respect to the information fed back by layout. However, a different approach completely remaps the entire circuit based on the information fed back from the layout design, making it difficult to guarantee any convergence when performing the iteration between layout and synthesis.

Another conventional approach is to use a concurrent design flow, which performs logic synthesis/technology mapping and placement/routing concurrently. However, the optimality of such an approach usually holds for very special circuit structures (such as trees) and the main difficulty associated with this approach is its high computational complexity.

Clearly, a better technique is needed for an effective and efficient layout-driven synthesis flow. Such a technique should consider layout information during synthesis and design planning, such as by combining logic partitioning with retiming and proper consideration of global and local interconnect delays, or by exploiting fast interconnects available in many PLD architectures during technology mapping.

As the capacity of PLD devices increases, hierarchical architectures are being more widely used, where basic programmable logic blocks, such as look-up tables (LUTs) or macrocells, are grouped into a logic cluster and connected by local programmable interconnects inside the cluster. There are basically two types of clusters, hard-wired connection-based clusters (HCC) and programmable interconnect-based clusters (PIC). The layout-driven synthesis flow described below is mainly targeted for the PIC-based FPGA architectures, although, in other embodiments, other architectures are targeted.

Please amend paragraph [0002] on page 1 as follows:

[0002] In a ~~programmable interconnect based cluster (~~PIC~~)~~, a group of basic logic blocks are connected by a local programmable interconnection array that usually provides full connectivity and is much faster than global or semi-global programmable interconnects. A number of commercial PLDs use the PIC architecture, such as the logic array block (LAB) in Altera FLEX 10K and APEX 20K devices, and the MegaLAB in APEX 20K devices. For example, in FLEX 10K devices, each LAB consists of eight 4-LUTs connected by the local interconnect array. Multi-level hierarchy can be formed easily using PICs, in which a group of small (lower-level) PICs may be connected through a programmable interconnect array at this level to form a larger (higher-level) PIC. For example, in Altera APEX 20K FPGAs, each LAB consists of ten 4-LUTs connected by local interconnects, which forms the first-level PIC. Then,

16 such LABs, together with one embedded system block and another level of programmable interconnects, form a second level PIC, called MegaLAB. Finally, global interconnects are used to route between MegaLAB structures and to I/O pins.

Please add the following paragraph after paragraph [0002] on page 1:

Conventional PLD synthesis algorithms often transform a given design into a flat netlist of basic programmable logic blocks (such as LUTs or macrocells) without consideration of the device hierarchy. Therefore, a substantial challenge in this area is to be able to synthesize a given design directly into a multi-level hierarchical architecture, with consideration of different interconnect delays and clustering constraints at each level.

Please delete paragraphs [0003] to [0007] on pages 1-3.

Please amend paragraph [0010] beginning on page 3 as follows:

[0010] Embodiments incorporating the novel methodology for the general timing-driven iterative refinement-based approach have some or all of the following characteristics and advantages:

1. The need for the (incremental) timing analysis during the iterative refinement procedure is reduced or eliminated. ~~completely eliminated. Depending on how often the timing analysis is invoked in order to update the timing at each node, the time spent on timing analysis could be a significant portion with respect to the time spent on the entire resynthesis process. Thus, the elimination of the need to perform timing analysis can significantly improve the efficiency of the resynthesis procedure.~~

2. The local transformation is able to see and use more accurate timing information (e.g., the arrival times at the transitive fanin signals) so that the transformations can be conducted in a more meaningful way to reduce the circuit delay.
3. Design preferences can be much more easily considered because of the flexibility of the methodology (e.g., in hybrid FPGAs with both LUT clusters and Pterm blocks, it is better to use the same logic resources consecutively on a critical path so that the subsequent clustering procedure can pack these implementations into one cluster to reduce the circuit delay).
4. A general framework is provided which allows the integration of several types of local transformations, such as logic resynthesis, mapping, clustering, and so on, to enable an integration of currently separated design processes.

Please delete paragraph [0012] on page 4.

Please amend paragraph [0024] beginning on page 5 as follows:

[0024] A novel methodology or process of timing optimization based on iterative refinement will be described. Use of this novel methodology reduces or ~~can~~ eliminates the need for (incremental) timing analysis during the iterative refinement procedure and any local transformation is able to utilize the more accurate timing information from the recursive delay reduction process described below. The timing-driven logic resynthesis or synthesis methodology utilizes a delay reduction process to reduce the delay of node v. The delay reduction process attempts to recursively reduce the delay of the critical fanins of node v instead of conducting the local transformation for v directly. Furthermore, in one embodiment, the fanins of node v are sorted in non-ascending order according to their slack values in non-ascending order. Thus, the fanins that have bigger negative slack values and hence are easier to speed up are processed before those fanins that have smaller negative slack values and hence are

more difficult to speed up. The novel optimization methodology is able to outperform the state-of-the-art algorithms consistently while significantly reducing the run time.

Please amend paragraph [0036] on page 8 as follows:

[0036] The *required time* at the output of every node v, denoted as $\bar{t}(v)$, is the required arrival time at v in order to meet the overall circuit timing constraint ~~as defined by the circuit designer or by the automatic optimization routine~~. The required times may be propagated backwards, from the POs to the PIs, by way of a backward network traversal. Let u_i be the ith fanout node of v, and v be the jth fanin of node u_i, then:

$$\bar{t}(v) = \min_{0 \le i < |output(v)|} (\bar{t}(u_i) - d_j(u_i) - d(v,u_i)) \quad (2)$$

Please amend paragraph [0058] on page 16 as follows:

[0058] A feature of the *reduceDelay* process 708 is that in order to reduce the delay for node v, instead of conducting the local transformation for v directly, the process recursively reduces the delay of the critical fanins of v, where possible. It is clear that in this procedure, the transformations ~~The critical fanins of the node v are~~ closer to the primary inputs ~~compared to the node v itself. If the reduceDelay process 708 at those critical fanins can result in node v meeting its delay target, then node v itself does not have to go through a local transformation for delay reduction. Therefore, the local transformations closer to the primary inputs~~ are ~~automatically~~ preferred ~~if they can help meet the overall delay targets~~.

Please amend paragraph [0063] on page 17 as follows:

[0063] The novel methodology for the general timing-driven iterative refinement-based approach has the following advantages, though particular embodiments may include only some of the advantages:

1. It reduces or completely eliminates the need for the (incremental) timing analysis during the iterative refinement procedure.
2. It allows the local transformation to be able to see more accurate timing information (e.g., the arrival times at the transitive fanin signals) so that the transformations may be conducted in a more meaningful way to reduce the circuit delay.
3. Its flexibility makes it much easier to consider the design preferences (e.g., in hybrid FPGAs with both LUT clusters and Pterm blocks, it is better to use the same logic resources consecutively on a critical path so that the subsequent clustering procedure may pack these implementations into one cluster to reduce the circuit delay.)
4. It provides a general framework to integrate several types of local transformations, such as logic resynthesis, mapping, clustering, and so on, to enable an integration of currently separate design processes.

Please delete paragraph [0077] on page 23.

Please amend paragraph [0096] on page 29 as follows:

[0096] A further interest is to ~~To~~ understand the impact of the timing optimization on the circuit delay after the layout design. Although the timing optimization is very effective in optimizing the circuit depth during the technology independent stage, it may cause the circuit delay to increase if the layout effect is ignored during optimization. Therefore, the layout-driven

synthesis flow~~, a~~ was described above. A comparison of the area optimization (opt_{area}), timing optimization (*TDO*) and clustering-driven timing optimization (*layoutDrivenTDO*) was performed. Although there is no dramatic delay reduction, *layoutDrivenTDO* is able to achieve better area and better delay results compared to *TDO* which does not consider the layout effect.

Please amend paragraph [0098] on page 29 as follows:

[0098] The novel methodology for the general timing-driven iterative refinement-based approach has the following characteristics and advantages:

1. It reduces or eliminates the need for the (incremental) timing analysis during the iterative refinement procedure.
2. It allows the local transformation to be able to see more accurate timing information (e.g., the arrival times at the transitive fanin signals) so that the transformations can be conducted in a more meaningful way to reduce the circuit delay.
3. Its flexibility makes it much easier to consider the design preferences (e.g., in hybrid FPGAs with both LUT clusters and Pterm blocks, it is better to use the same logic resources consecutively on a critical path so that the subsequent clustering procedure can pack these implementations into one cluster to reduce the circuit delay.)
4. It provides a general framework to integrate several types of local transformations, such as logic resynthesis, mapping, clustering, and so on, to enable an integration of currently separated design processes.